PLATAFORMA DE INTEGRAÇÃO COM REDE DAS COISAS (WEB OF THINGS)

Rúben Vera-Cruz Quintas

Faculdade de Ciências, Universidade de Lisboa

30.07.2010

AGRADECIMENTOS

Gostaria de deixar o meu agradecimento ao meu tutor, Professor Doutor Francisco Martins, da Faculdade de Ciências da Universidade de Lisboa, pelo apoio e dedicação dispendidos ao longo deste ano e do acompanhamento do projecto de investigação.

Índice

Enquadramento e Objectivos

A *Internet* veio revolucionar a forma como as pessoas interagem com o mundo que as rodeia. Hoje em dia, é comum as pessoas encontrarem-se à volta de um *chat*, lerem os *blogs* uns dos outros, fazerem compras sem sair da sua secretária, etc. Para tal contribuiu a *web of services*, que permite oferecer produtos e serviços num mundo virtual a clientes também virtuais. No entanto, falta a este mundo a ligação com o estado real das coisas. Por exemplo, numa aplicação de Domótica (vulgo casas inteligentes) seria importante conhecer a temperatura a que o frigorífico se encontra ou a luminosidade da sala de estar.

Neste projecto propomos desenvolver uma plataforma intermédia (*middleware*) que permita publicar o estado de objectos reais a aplicações de alto nível. Em particular, pretende-se utilizar redes de sensores para captar informação do meio ambiente e torna-la acessível, de forma fácil e uniforme, a aplicações de alto nível.

O objectivo com esta plataforma é, por um lado interagir e actuar sobre uma rede de sensores de modo a monitorizar e controlar condições ambientais de objectos reais (temperatura, entre outros)[9] e, por outro, disponibilizar esta informação a uma camada superior que possa ser utilizada por aplicações de alto nível.

Actividades Desenvolvidas

Este projecto foi dividido em dois níveis. Um nível mais baixo que consiste na programação de redes de sensores para ler e reagir a estímulos externos, como a temperatura, por exemplo. Esta camada permite a adaptação dos serviços dos sensores consoante as necessidades dos clientes. O outro nível contemplado neste projecto é um nível intermédio, *middleware*, que suporta os protocolos de *publish/subscribe*[1] para responder aos pedidos feitos por uma *Web Application*.[2]

Baixo Nível

Aprendizagem de uma linguagem de Programação de redes de Sensores. Uma rede de sensores é composta por vários nós (cada nó corresponde a um sensor), que notificam a rede periodicamente com os valores lidos do ambiente externo.

Devido à complexidade no desenvolvimento das redes, e também à forma como as linguagens de programação actuais para estas plataformas são desenvolvidas, existe alguma dificuldade em programar estas redes e, consequentemente, em fazer depuração de erros de aplicação para sensores. Por outro lado, as implementações actuais não se esforçam o suficiente em fornecer descrições formais da linguagem e do seu sistema de execução, nem em provar propriedades estáticas, como por exemplo, a segurança da memória.

Para tal, neste nível foi usada uma linguagem de programação para redes de sensores, a linguagem *Callas*[6][7], que permite mobilidade e actualização de código e vai de encontro às limitações enumeradas acima.

1 – Este protocolo é vantajoso na medida em que a parte do cliente (*Subscriber*) não precisa de verificar quando é que o evento é desencadeado. Estando registado para um evento, quando o *middleware* (*Publisher*) verifica que esse evento é desencadeado, notifica o cliente.

2 – Esta *Web Application* foi desenvolvida por outra colega, Ana Lúcia Rodrigues, a quem também foi atribuída uma bolsa da Fundação Amadeu Dias e a cargo do mesmo tutor. O nome da aplicação é Sistema de Monitorização e Controlo Museológico e interage com este projecto.

Na figura que se segue é apresentado um excerto de código que faz com que, periodicamente, seja transmitida uma notificação sobre a temperatura ambiente de um nó da rede. Estas notificações são enviadas de 10 em 10 minutos durante uma semana.

```
1  defmodule Nil:
2    pass
3
4  defmodule Sampler:
5    Nil sample()              # sample the temperature
6  # declare module sampler, install it, and call sample() periodically
7  module s of Sampler:
8    def sample(self):
9      # assign the sensed temperature to 'curTemp'
10     curTemp = extern getTemp()
11     send log(curTemp)
12
13 mem = load                  # load the sensor memory
14 newMem = mem || s           # update function sample
15 store newMem                # replace the sensor memory
16 # invoke sample() every ten minutes, for one week
17 sample() every 60*10 expire 60*60*24*7
```

Fig. 1 - Programa para, periodicamente, transmitir a temperatura ambiente de um nó para a rede.

O programa obtém a temperatura do ambiente executando a instrução na linha 10.
A comunicação feita com a rede acontece na linha 11 através do comando *send log(curTemp).* Este comando envia a temperatura para os nós vizinhos que por sua vez encaminham a mensagem até à estação central.
Na linha 17 é definido um temporizador que irá proceder à leitura, periodicamente, dos valores do ambiente:

"**every** 60*10" – 60 segundos (1 minuto) x 10 = 10 minutos;

"**expire** 60*60*24*7" – 60 segundos (1 minuto) x 60 = 60 minutos (1hora) x 24 = 24 horas (1 dia) x 7 = 7 dias (1 semana).

Para melhor familiarização com a linguagem, procedi à implementação de diversos exemplos e à sua execução em sensores *SunSpot*[4].
Além disso aprendi a trabalhar na plataforma de desenvolvimento *Sun* para os sensores que utilizámos.

Nível Intermédio

Este nível do sistema permite a reprogramação dos sensores conforme os requisitos do cliente, bem como a interacção com a rede de sensores. A reprogramação é efectuada através de invocações remotas de funções na rede. O código que corre nos sensores não é executado no mesmo ambiente da aplicação de *middleware*. A forma de interagir com este código faz-se por chamada remota a métodos, utilizando a tecnologia *RMI* (*Remote Method Invocation*), facilitando a integração de novo código na rede de sensores.

Para essa interacção com a rede de sensores, é usada novamente a linguagem de programação *Callas*, mencionada antes.

Esta camada do projecto é composta por dois componentes, um que funciona como um Observador e outro que funciona como o Observável. Estes dois componentes formam o padrão *Observer*, especificado nos "Métodos Escolhidos", onde o Observador fica à espera que certos eventos sejam notificados pelo Observável.

A aplicação que desenvolvi abstrai a rede de sensores e permite que aplicações de alto nível leiam valores do ambiente (através de sensores) sem que tenham de ter que lidar com a complexidade de interagir com os sensores directamente.

Métodos Escolhidos

No desenvolvimento do projecto foram tomadas várias decisões com o intuito de simplificar a comunicação entre o *middleware* e a rede de sensores.

Foram seguidos padrões de desenvolvimento de *software* mencionados na bibliografia[1][2].

Arquitectura

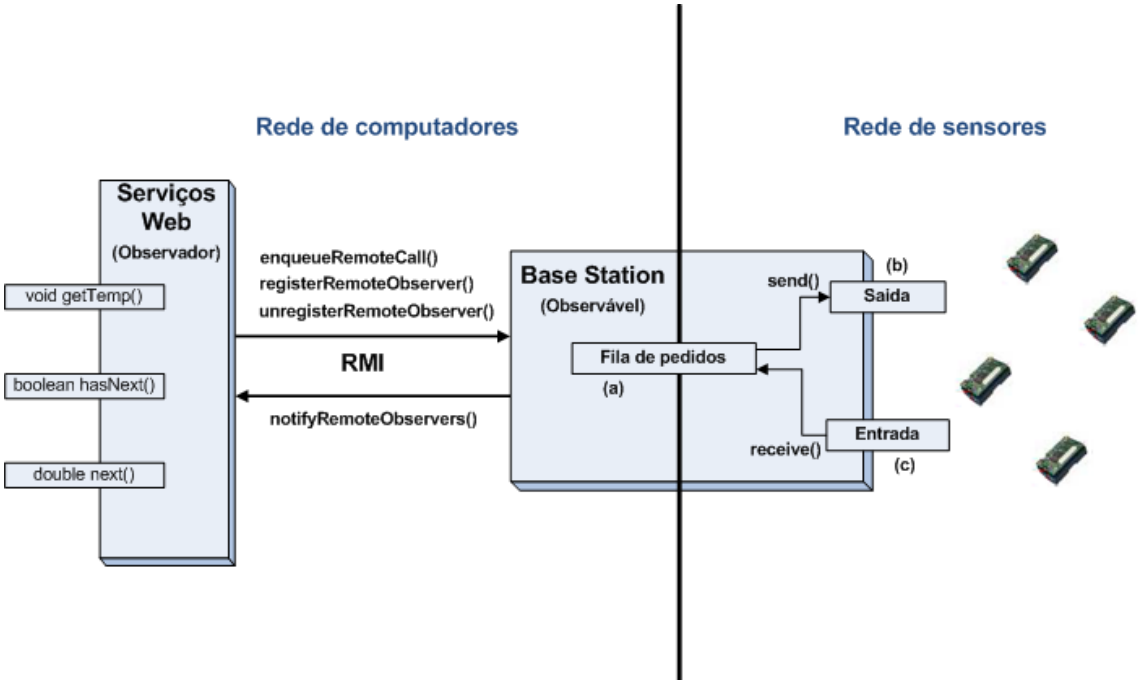


Fig. 2 – Arquitectura do sistema

A Fig. 2 dá uma visão global do sistema, que junta duas redes distintas: uma rede de computadores e uma rede de sensores. Estas duas redes comunicam entre si através de um nó comum, identificado na figura como *Base Station* (BS).

O BS executa uma máquina virtual (CVM) que corre o código compilado da linguagem *Callas* e que comunica com a rede de sensores através de duas filas: uma de saída (b) e outra de entrada (c). Estas filas são manipuladas utilizando os comandos *send* e *receive* da linguagem *Callas*. Como o *hardware* dos sensores é muito limitado, entre outros, em recursos de computação, a CVM recorre a uma fila de pedidos (a) para registar acções pendentes, em vez de recorrer a fios de execução.

O cliente do serviço *web* acede à rede de sensores através da interacção destes com o BS, que é responsável por encaminhar os pedidos para a rede de sensores, recolher as respostas e entregá-las de volta ao serviço *web*.

Quando a rede de sensores responde à invocação de um serviço, o BS notifica o serviço *web*, enviando uma lista de resultados. A partir do momento em que o serviço *web* recebe a resposta ao pedido fica responsável pela gestão dos resultados, tendo o cliente que invocar os restantes métodos que lhe são disponibilizados para ter acesso aos resultados.

A camada de cliente comunica com o *middleware* através dos serviços ou métodos disponibilizados pelo serviço *web*.

Tecnologias

RMI - Esta tecnologia permite que seja possível invocar métodos remotamente. Esta mesma tecnologia é usada na comunicação entre os serviços *web* e o BS do sistema na reprogramação dos sensores e da rede.

Sensores - *SunSpot*[4] e *Arduino*[5].

Ferramentas

Netbeans - Software[8] usado na parte do desenvolvimento em linguagem *Java*[3].

Padrões Utilizados

Remote Observer - Este padrão permite que uma aplicação cliente, executada remotamente, seja notificada quando ocorre um evento no qual se registou, como numa arquitectura *publish*/*subscribe*.
É constituído por um conjunto de observadores (no nosso caso os serviços *web*) registados num observável (a *Base Station*). Este, por sua vez, é responsável por notificar os observadores que se registam no observável para algum evento, quando esse mesmo evento é desencadeado.
O BS permite ainda, que o mesmo observador esteja registado nos vários serviços que a rede de sensores disponibiliza. Para que tal seja possível o observador envia, na altura do registo, um identificador e

uma etiqueta que representa o serviço no qual pretende registar-se.

Adapter - Este padrão serve para resolver a incompatibilidades entre as interfaces dos componentes. O *Adapter* define a interface que tem de ser usada pelos componentes para ser realizada a comunicação.
É usado na comunicação entre o *middleware* e a rede de sensores e entre o *middleware* e a *Web Application.*

Factory - Este padrão é usado para, por exemplo, se obter os objectos A*dapter* necessários. Quando a camada de *middleware* quer comunicar com a rede de Sensores *SunSpot*, pede a uma *Factory* o A*dapter* correspondente a esses sensores. Caso queira comunicar com uma rede de Sensores *Arduino*, o *Factory* fornece a interface correspondente.

Singleton - Este padrão define que uma classe é instanciada apenas uma vez na execução do sistema. É o caso da *Factory* que apenas existe uma instância que pode ser usada em diversos sítios do sistema. Este padrão garante ainda que em qualquer sítio que seja necessário usar a *Factory* para obter alguma interface, é usada sempre a mesma instância, não havendo duplicação de recursos.

Iterator - Este padrão é usado para abstrair a leitura de informação e efectuar a leitura de modo uniforme às diversas redes.

Resultados Obtidos

Aplicação de middleware – A rede de Sensores programada neste projecto mede os valores de ambiente e envia-os para uma camada superior, a aplicação de *middleware*. Esta, por sua vez, disponibiliza a informação através de serviços *web* de maneira a fazer chegar essa informação à *Web Application*.

Prova de Conceito – De modo a demonstrar que o *software* desenvolvido é utilizável por outras aplicações que, por algum motivo, queiram ou necessitem de obter informações sobre o ambiente de uma rede de Sensores, interpôs-se este projecto com o projecto realizado pela minha colega Ana Rodrigues, que desenvolveu uma aplicação para conteúdo museológico.

Esta aplicação precisa de monitorizam a temperatura e a humidade de um dado ambiente e este projecto fornece essa mesma informação, possibilitando ainda possíveis reprogramações da rede de Sensores.

Execução Financeira

Durante o decorrer deste projecto foram gastos aproximadamente 1100€ para adquirir três *kits* de sensores *SunSpot* e dois sensores *Arduino*. Este valor foi suportado pelo projecto CALLAS. O resto dos recursos utilizados no projecto, como o *software*, foram obtidos recorrendo a *software* livre, não envolvendo assim quaisquer custos de licenciamento.

O posto de trabalho no laboratório *LaSIGE* foi suportado pelo projecto CALLAS.

Para além disso, estava proposto frequentar uma acção de formação sobre a conservação e monitorização museológica, ministrada pelo Instituto Ibérico de Património. Essa acção foi cancelada pelo instituto, ainda não tendo data prevista para a sua realização.

Pretendemos frequentar a formação quando for anunciada a nova data, na tentativa de aperfeiçoar e aproximar o projecto da realidade museológica.

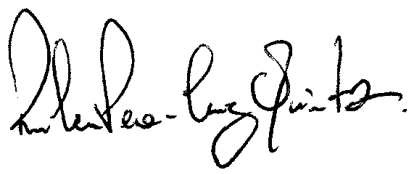
Conclusões

Com este projecto foi possível desenvolver um *middleware* capaz de comunicar com uma rede de sensores e, se necessário, possibilita a sua reprogramação, sem ser necessário reprogramar cada nó da rede. A rede comunica ao *middleware* o estado ou a informação do meio ambiente e este *middleware* faz chegar essa informação a uma *Web Application*, neste caso o "Sistema de Controlo Monitorização Museológica" da colega Ana Lúcia Rodrigues.

Pensa-se, depois ainda de algum aprimoramento, que este projecto possa ser algo viável a nível de mercado, principalmente a nível museológico, como foi proposto neste projecto, mas não só.

Para finalizar o projecto proporcionou um grande conjunto de novos conhecimentos, não só por tratar de assuntos não vistos durante a licenciatura, mas também por fortalecer outros já falados.

Lisboa, 30 de Julho, de 2010.



(Aluno) (Professor Doutor Francisco Martins)

Bibliografia

Livros:

[1]. *Applying UML and Patterns - An Introduction to Object-oriented Analysis and Design*, Craig Larman, Prentice-Hall, 2002, ISBN 0-13-092569-1;

[2]. *Design Patterns: Elements of Reusable Object-Oriented Software*, E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Addison-Wesley, ISBN 0-201-63361-2;

[3]. *The JAVA Programming Language*, Fourth Edition, K. Arnold, J.Gosling, D.Holmes, Addison-Wesley 2005, ISBN 0-321-34980-6;

Sensores SunSpot:

[4]. http://sunspotworld.com/

Sensores Arduino:

[5]. http://www.arduino.cc/

Linguagem Callas:

[6]. http://www.dcc.fc.up.pt/callas/publications/programming-WSN-minema-winter.pdf

[7]. http://www.dcc.fc.up.pt/callas/publications/silva.martins.etal-calculus-sensor-networks-arxiv.pdf

Software Netbeans:

[8]. http://netbeans.org/

Documento de monitorização museológica:

[9]. http://www.patrimoniocultural.org/demu/bibliografia/arquivos/monitoramento_temperatura.pdf